



**SUBSTITUTE SPECIFICATION**

**CLEAN VERSION – FOR ENTRY**

# REPRESENTATION OF OBJECTS IN A JAVA PROGRAMMING ENVIRONMENT

## BACKGROUND OF THE INVENTION

5           The present invention relates generally to object-based high level programming environments, and more particularly, to techniques suitable for representation of objects in a Java™ programming environment.

          One of the goals of high level languages is to provide a portable programming environment such that the computer programs may easily be  
10       ported to another computer platform. High level languages such as "C" provide a level of abstraction from the underlying computer architecture and their success is well evidenced from the fact that most computer applications are now written in a high level language.

          Portability has been taken to new heights with the advent of the  
15       World Wide Web ("the Web") which is an interface protocol for the Internet which allows communication between diverse computer platforms through a graphical interface. Computers communicating over the Web are able to download and execute small applications called applets. Given that applets may be executed on a diverse assortment of computer platforms,  
20       the applets are typically executed by a Java™ virtual machine.

          Recently, the Java™ programming environment has become quite popular. The Java™ programming language is a language that is designed to be portable enough to be executed on a wide range of computers ranging from small devices (e.g., pagers, cell phones and smart cards) up  
25       to supercomputers. Computer programs written in the Java™ programming language (and other languages) may be compiled into Java™ Bytecode instructions that are suitable for execution by a Java™ virtual machine implementation. The Java™ virtual machine is commonly implemented in software by means of an interpreter for the Java™ virtual machine  
30       instruction set but, in general, may be software, hardware, or both. A

particular Java™ virtual machine implementation and corresponding support libraries together constitute a Java™ runtime environment.

Computer programs in the Java™ programming language are arranged in one or more classes or interfaces (referred to herein jointly as  
5 classes or class files). Such programs are generally platform, i.e., hardware and operating system, independent. As such, these computer programs may be executed without modification on any computer that is able to run an implementation of the Java™ runtime environment.

Object-oriented classes written in the Java™ programming language  
10 are compiled to a particular binary format called the “class file format.” The class file includes various components associated with a single class. These components can be, for example, methods and/or interfaces associated with the class. In addition, the class file format can include a significant amount of ancillary information that is associated with the class.  
15 The class file format (as well as the general operation of the Java™ virtual machine) is described in some detail in The Java Virtual Machine Specification, Second Edition, by Tim Lindholm and Frank Yellin, which is hereby incorporated herein by reference.

Fig. 1A shows a progression of a simple piece of a Java™ source  
20 code 101 through execution by an interpreter, the Java™ virtual machine. The Java™ source code 101 includes the classic Hello World program written in Java™. The source code is then input into a Bytecode compiler 103 that compiles the source code into Bytecodes. The Bytecodes are virtual machine instructions as they will be executed by a software  
25 emulated computer. Typically, virtual machine instructions are generic (i.e., not designed for any specific microprocessor or computer architecture) but this is not required. The Bytecode compiler outputs a Java™ class file 105 that includes the Bytecodes for the Java™ program. The Java™ class file is input into a Java™ virtual machine 107. The Java™ virtual machine is an  
30 interpreter that decodes and executes the Bytecodes in the Java™ class file. The Java™ virtual machine is an interpreter, but is commonly referred to as a virtual machine as it emulates a microprocessor or computer

architecture in software (e.g., the microprocessor or computer architecture may not exist in hardware).

Fig. 1B illustrates a simplified class file 100. As shown in Fig. 1B, the class file 100 includes a constant pool 102 portion, interfaces portion 104, fields portion 106, methods portion 108, and attributes portion 110. The attributes (or attributes table) 110 portion represents the attributes associated with the class file 100. This allows for one or more attributes to be defined, each of which can be associated with one or more components of the class file. As is known to those skilled in the art, the Java™ virtual machine implementations are allowed to define and use various attributes. In addition, the virtual machine's implementations ignore attributes that they do not recognize. Thus, a class file may contain one or more attributes, all or none of which may be recognized by a particular virtual machine implementation.

Conventionally, Java™ objects are represented in memory so that the methods associated with the objects can be referenced from the object representation. Typically, there is a reference from the Java™ object representation directly to a method table that includes the methods associated with the object. Although the direct reference to the method table allows method invocations to be performed, the conventional object representation in Java™ requires some processing to find information about the object (e.g., object type, object size, static fields, etc.) Such information about the Java™ object can be stored in the internal class representation of the object. In other words, the virtual machine typically internally represents and stores the information associated with the Java™ object's class. However, accessing this information takes up valuable processing time. This can seriously hinder performance of virtual machines, especially in systems with limited computing power and/or memory.

In view of the foregoing, improved techniques for representation of objects in Java™ programming environments are needed.

## **SUMMARY OF THE INVENTION**

Broadly speaking, the present invention relates to techniques for representation of objects in a Java™ programming environment. The techniques are highly suitable for representation of Java™ objects inside virtual machines, especially those that operate with limited resources (e.g., embedded systems). In accordance with one aspect of the invention, a Java™ object representation is disclosed. As will be appreciated, the Java™ object representation provides a reference that can be used to directly access the internal class representation associated with the object. The internal class representation provides information regarding the Java™ object (e.g., object size, object type, static fields, etc.) As a result, the invention allows quick access to information regarding Java™ objects. This means that the processing time conventionally needed to access information regarding Java™ objects is reduced. Thus, the invention can enhance performance of virtual machines, especially in systems with limited computing power and/or memory.

The invention can be implemented in numerous ways, including as a method, an apparatus, a computer readable medium, and a database system. Several embodiments of the invention are discussed below.

As a Java™ object representation suitable for use by a Java™ virtual machine, one embodiment of the invention includes a first reference to an internal class representation of the Java™ object, a second reference to instance fields associated with the Java™ object. The first reference is a direct reference to the internal class representation of the Java™ object.

As a method for representing a Java™ object in a virtual machine, one embodiment of the invention includes the acts of: allocating a first reference in a memory portion of the virtual machine, wherein the first reference is a reference to an internal class representation of the Java™ object; and allocating a second reference in a memory portion of the virtual machine, wherein the second reference is a reference to instance fields

associated with the Java™ object; and wherein the first reference is a direct reference to the internal class representation of the Java™ object.

As a method of accessing information regarding a Java™ object, one embodiment of the invention includes the acts of identifying an object representation associated with the Java™ object; using a first reference in the object representation to locate an appropriate internal class representation associated with the Java™ object; accessing information regarding the Java™ object from the internal class representation; and wherein the object is represented in a Java™ virtual machine.

As a computer readable media including computer program code for a Java™ object representation suitable for use by a Java™ virtual machine, one embodiment of the invention includes computer program code for a first reference to an internal class representation of the Java™ object; computer program code for a second reference to instance fields associated with the Java™ object. The first reference is a direct reference to the internal class representation of the Java™ object.

These and other aspects and advantages of the present invention will become more apparent when the detailed description below is read in conjunction with the accompanying drawings.

#### **BRIEF DESCRIPTION OF THE DRAWINGS**

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Fig. 1A shows a progression of a simple piece of a Java™ source code through execution by an interpreter, the Java™ virtual machine.

Fig. 1B illustrates a simplified class file.

Fig. 2 represents a Java™ computing environment including a Java™ object representation in accordance with one embodiment of the invention.

Fig 3 illustrates a method for representing Java™ objects in a Java™ computing environment.

Fig. 4 illustrates a method for accessing information regarding a Java™ object using an object representation, in accordance with one embodiment of the invention.

## **DETAILED DESCRIPTION OF THE INVENTION**

As noted in the background, typically, the virtual machines internally represent and store the information associated with the Java™ object's class. However, accessing this information using conventional techniques takes up valuable processing time. This can seriously hinder performance of virtual machines, especially in systems with limited computing power and/or memory.

The present invention pertains to techniques for representation of objects in a Java™ programming environment. The techniques are highly suitable for representation of Java™ objects inside virtual machines, especially those that operate with limited resources (e.g., embedded systems). In accordance with one aspect of the invention, a Java™ object representation is disclosed. As will be appreciated, the Java™ object representation provides a reference that can be used to directly access the internal class representation associated with the object. The internal class representation provides information regarding the Java™ object (e.g., object size, object type, static fields, etc.) As a result, the invention allows quick access to information regarding Java™ objects. This means that the processing time conventionally needed to access information regarding Java™ objects is reduced. Thus, the invention can enhance performance of virtual machines, especially in systems with limited computing power and/or memory.

Embodiments of the invention are discussed below with reference to Figs. 2-4. However, those skilled in the art will readily appreciate that the detailed description given herein with respect to these figures is for explanatory purposes only as the invention extends beyond these limited  
5      embodiments.

Fig. 2 represents a Java™ computing environment including a Java™ object representation 200 in accordance with one embodiment of the invention. The object representation 200 is suitable for representation of a Java™ object in a virtual machine. As shown in Fig. 2, the object  
10      representation 200 includes a first reference 202 to an internal class representation 204. The internal class representation 204 provides information regarding the Java™ object. This information can include, for example, a method table 206 and a field descriptor table 208, as well as other information relating to the Java™ object. In the described  
15      embodiment, the method table 210 immediately follows a header 209 which is of a predetermined size.

As will be appreciated, the first reference 202 can be used to directly access the internal class representation 204 so that information regarding the Java™ object can be accessed quickly. As a result, information  
20      regarding objects can be accessed more quickly than conventional techniques which require more processing to find this information.

In addition, the object representation 200 includes a second reference 210 to instance fields associated with the Java™ object. These instance fields can be unique for each object and can, for example, include  
25      instance variables associated with the Java™ object. Instance fields in the context of the Java™ programming language are well known to those skilled in the art.

It should be noted that the internal object representation 200 may include an identifier that uniquely identifies the Java™ object. As will be  
30      appreciated by those skilled in the art, the identifier can be a hash key. In one embodiment, the address of the first reference 202 is used as the hash

key. It should also be noted that the first and second references 202 and 210 represent two consecutive memory addresses. As such, each of the first and second references 202 and 210 can be four consecutive bytes (one word) in a memory portion of the virtual machine.

5           Fig 3 illustrates a method 300 for representing Java™ objects in a Java™ computing environment. Method 300 can be used by a Java™ virtual machine to represent Java™ objects. Initially, at operation 302, two consecutive words is allocated in memory. Four consecutive bytes can be allocated for each word which can be used to represent a reference (e.g., a  
10   pointer to another memory location). Next, at operation 304, the first reference (i.e., first word) is set to the location of an internal class representation for the class which the Java™ object belongs to. Thereafter, at operation 306, the second reference is set to the location of the instance fields associated with the Java™ object. Finally, at operation 308, the  
15   address of the first reference is stored. As will be appreciated, this address can be used as a hash key to identify and/or locate the representation of the Java™ object.

          Fig. 4 illustrates a method 400 for accessing information regarding a Java™ object using an object representation, in accordance with one  
20   embodiment of the invention. The method 400 can be used by a virtual machine to access information regarding the Java™ object. Initially, at operation 402, the appropriate hash key is used to locate the representation of the Java™ object. Next, at operation 404, the first reference of the Java™ object representation is used to locate the  
25   appropriate internal class representation for the Java™ object. Thereafter, at operation 406, a determination is made as to whether a method associated with the object needs to be invoked. If it is determined at operation 406 that a method associated with the object needs to be invoked, the method 400 proceeds to operation 408 where the header  
30   associated with the internal class representation is skipped. Accordingly, the method table that follows the header is accessed at operation 410 to invoke the appropriate method. The method 400 ends following operation

410. However, if it is determined at operation 406 that a method associated with the object does not need to be invoked, the method 400 proceeds to operation 412 where the information regarding the Java™ object can be accessed from appropriate entries in the internal class representation.

The many features and advantages of the present invention are apparent from the written description, and thus, it is intended by the appended claims to cover all such features and advantages of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and operation as illustrated and described. Hence, all suitable modifications and equivalents may be resorted to as falling within the scope of the invention.

*What is claimed is:*